IN THE SPECIFICATION:

1. Please replace the paragraph that begins on page 3, line 13, with the following paragraph:

To generate the verification model, source code parsing techniques are use to build a control flow graph for procedural elements of the source code program. The control flow graph is expressed in alternate specification languages (*target* languages), including the one that is used by the logic model checking tool SPIN. The basic statements from the source code can be adjusted to be expressible in the target language. The basic statements are considered relevant if the validity of the properties to be checked can be determined to be dependent thereon. In some cases they can be included as is (if the semantics of the target language allow this), in some cases they can be omitted from the generated model (if the validity of the properties to be checked can be determined to be independent of the statements to be dropped *i.e.*, irrelevant) and in some cases they can be syntactically converted in a format that is compatible with the target language (an equivalent statement). Model extraction according to the principles of the invention makes these determinations and can generate a syntactically correct model in the target language without user intervention.

2. Please replace the paragraph that begins on page10, line 10, with the following paragraph:

Optionally, the control flow graph for the target model that is constructed in the manner described above can be simplified prior to generation of the final verification model. In a decision step 28, it is determined whether the parse tree will be simplified. If the simplification is selected, the parse tree is simplified in a step 30 by recursively removing nodes corresponding to null ~~nul~~ statements, by removing the successors of false nodes, and by skipping true nodes under certain instances. For example:

3. Please replace the paragraph that begins on page11, line 23, with the following paragraph:

The optional routine 50 shown in FIG. 2 provides information about the completeness of the conversion table. The routine 50 begins with a parse tree in a step 52. Definition and use information (information related to the use of the data object in an expression) for each data object is collected at each marked node in the tree, recursively, as at 54. The information is stored at the marked node in three-linked lists with pointers to the symbol table for data, as at 56. The three linked lists are for data objects that are used, data objects that are defined, and data objects that are accessed in a manner that cannot easily be determined. For example, access to data via pointers is considered to be data that is accessed in a manner that cannot easily be determined. A data dependency graph based on the definition and use information is constructed in a step 58. Nodes in a first graph, DG, correspond to distinct data objects. For example, in DG there is a directed edge from node X to node Y if data object Y is used at least once in a definition of data object X. The transitive closure for the dependency relation is computed for DG, and additional edges are added to DG in accordance with the computation. A copy of the data graph is made and given a new name, such as DG*.

4. Please replace the paragraph that begins on page14, line 21, with the following paragraph:

(D) Exemplary Systems

A block diagram for an exemplary system 100 is shown in FIG. 3. The system verification engine 114 implements model checking using, for example, the SPIN model checker described above. The system source code 108 is an input to the abstraction filter (i.e., lookup table) 110, which produces the verification model 112 according to the principles of the invention. The processes described with reference to FIGs. 1 and 2 are exemplary processes for providing the verification

model 112. A ~~The~~ system requirement ~~requirements~~ or property is ~~properties are~~ another input to the system 100, and can be described independently by conventional means as described earlier. The negation of the system requirement 104 is taken ~~104~~ to provide a formalization of all violating executions that can potentially exist 106. These potential violations 106 are checked against the model 112 in the verification engine 114.

5. Please replace the paragraph that begins on page15, line 9, with the following paragraph:

(E) Exemplary Feature Verifications

In accordance with an embodiment of the invention, feature verification is achieved in the context of call processing environments as discussed in greater detail in Appendix A hereto. In accordance with a further embodiment of the invention, a model extraction is performed as a function of ANSI C program code, e.g., a communications protocol implementation, i.e., an alternating bit protocol, as more fully set forth in Appendix B. Of course, the embodiments detailed herein are illustrative and not exhaustive. That is, the aspects of the invention can be applied to any source programing language and extract a verification model in the specification language of any suitable model checking system. In context of software verification systems the aspects of the invention are universally applied to any given source language, e.g., C, C++ or Java™ (a trademark of Sun Microsystems, Inc. in the United States and other countries) programming language ~~Java~~, and any given target language, e.g., the Spin model checker.